DOCUMENT RESUME

ED 053 533 EM 009 051

AUTHOR Boessenroth, Theodore; And Others

TITLE Engineering Operational CAI.

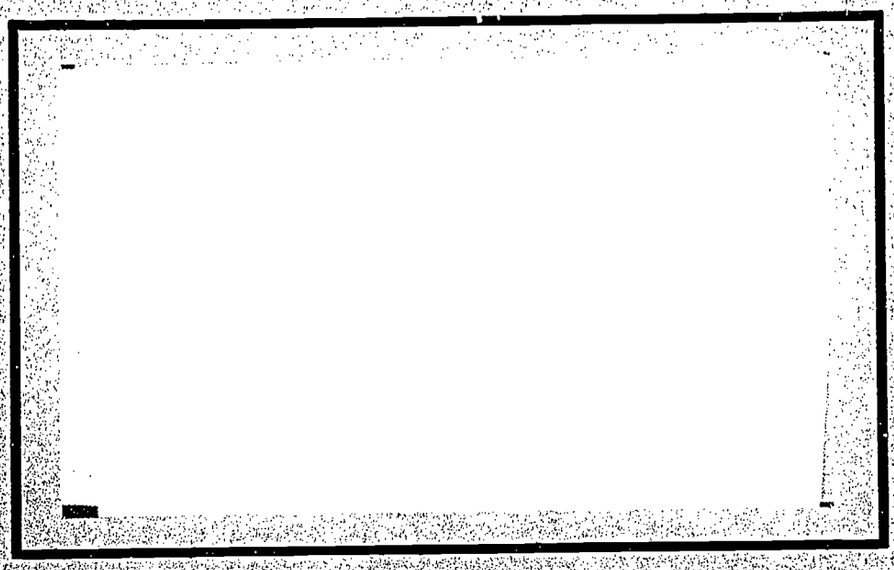
INSTITUTION Texas Univ., Austin. Computer-Assisted Instruction Lab.

SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO TM-1
PUB DATE Oct 70
NOTE 24p.

EDRS PRICE EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS *Communication Problems, *Computer Assisted Instruction, Curriculum Development, *Flowcharts, Programers, *Programing, Programing Languages, *Programing Problems

ABSTRACT

If a course using computer-assisted instruction (CAI) is to follow the author's philosophy and design, it is important that communication between author and coder be explicit. Here, a set of definitions and flowcharts are presented which allow an author to describe precisely to the coder alternate choices which a student using the program may make. By using these sample definitions and flowcharts, it is possible for author and coder to communicate clearly. The second part of this document describes an effective collaboration between author and coder in developing a course in mathematics fundamentals. Close contact between them produced consistency of approach and product. However, turnover of staff and deadline pressure made it difficult to maintain this close contact. The staff nevertheless felt such contact is desirable. (JK)



THE UNIVERSITY OF TEXAS AT AUSTIN

Computer Assisted Instruction Laboratory

AUSTIN

E N G I N E E R I N G

O P E R A T I O N A L C A I

TECHNICAL MEMO NO. 1

Theodore Boessenroth
Authella Smith
Carl Gregory

Sponsored By

THE NATIONAL SCIENCE FOUNDATION
Grant GJ 509 X

Computer-Assisted Instruction Laboratory
The University of Texas
Austin, Texas 78712

October 1970

ENGINEERING OPERATIONAL CAI

The instructor who considers writing a computer-assisted instruction (CAI) program first decides whether his material is suitable for CAI treatment. He discusses types of hardware available with the engineer and instructional designer and, in consultation with educational specialists, an instructional designer and programmer assess its potential. If he decides to use CAI, he begins course design with the preparation of terminal objectives: tasks the learner is to be capable of performing at the end of the course. Differences between classroom-lecture design and CAI design become apparent here; the former stresses stimulus dimensions, such as choice of text, behavior of the lecturer, continuity and elegance of delivery; the latter emphasizes response dimensions such as performance proficiency of the learner at the end of the course.

The author subsequently decides on prerequisite skills required and prepares intermediate objectives, a collection of intermediate performance requirements leading from the prerequisites to the terminal objectives, not necessarily linearly. Therefore, CAI design is a "backward" construction--terminal objectives to prerequisites--while lecture construction is forward--prerequisites to final lecture. Construction of intermediate objectives leads in a natural way to a modular structure of the CAI course, each module representing an instructional unit taking the learner from one or more intermediate objectives to another.

Writing of the course proper begins with construction of a skeleton for each module which is then refined to final form by off- and on-line testing using student feedback for continual revision. The module design again differs markedly from lecture preparation. The teacher designs his lectures based on verbal presentation with ample use of blackboard and text. Students' response requirements then are usually determined by what has been covered in class. Good CAI design is built on response requirements. The author must adapt himself to tight, efficient design exploiting display techniques of the new medium. The material has to be presented in small doses, one at a time, cognizant of the learners limited access to peripheral material.

The Author's Draft

Abstract

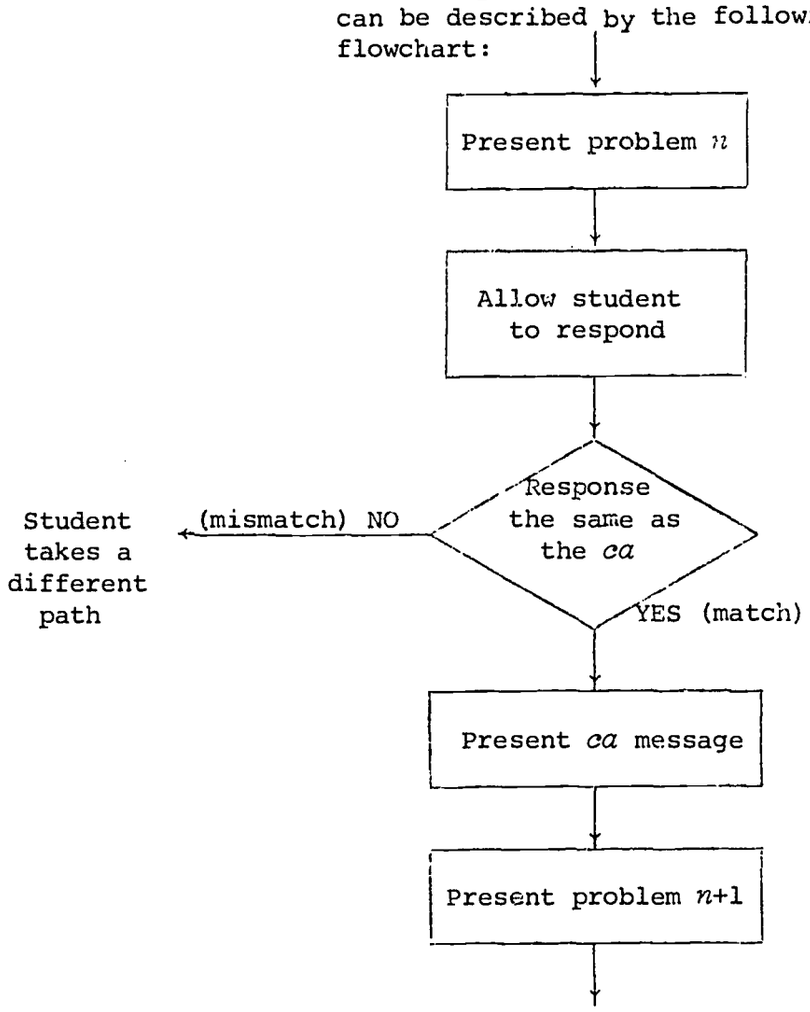
Designing a complex instructional system is difficult. Explaining that system, in detail, so that others will envision it in the same way as the designer is frustrating, if not impossible. So how, then, does an author communicate, accurately, with his coding staff? Certainly, if a coder could see, literally, the instructional system operate, then he could envision that instructional system in the same way as the designer. Thus, a design for an author's draft was developed which would do just that: permit the coder as well as the author to literally see the instructional system operate.

To prepare an author's draft of this design, the author needs pads of IBM 1510 Instructional Display Planning Guides, form X26-5608-0 (U/M 025), or facsimile, pencils, a pair of scissors, and a few well understood terms.

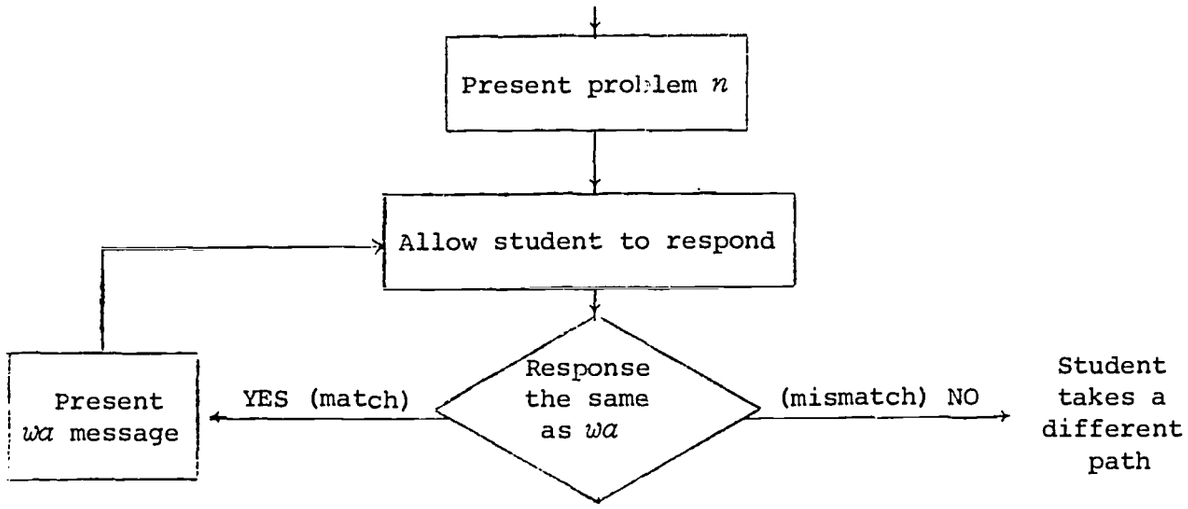
It is very important for the author to do his writing directly on a display planning guide so that he does not accidentally exceed the character limit of his particular display device--in our case, an IBM 1510 cathode ray tube (CRT). Thus, the author himself adjusts the text to fit into whatever display space is available. If there is no limit to display space, display guides are still useful to assist the author in developing some economy of language.

An author will usually have in mind a certain action he wants the computer to take dependent on the action a student has taken. He also will most likely want to study, at his leisure, student-computer interactions in order to determine the correctness, with respect to his intent, of the computer program; effectiveness of the instruction; clarity of messages; and ways to improve effectiveness. Thus, the author needs an easy way to specify dependent actions and identify what actions have occurred or are occurring for particular students or groups of students. The terms we are using for author-coder communication are as follows:

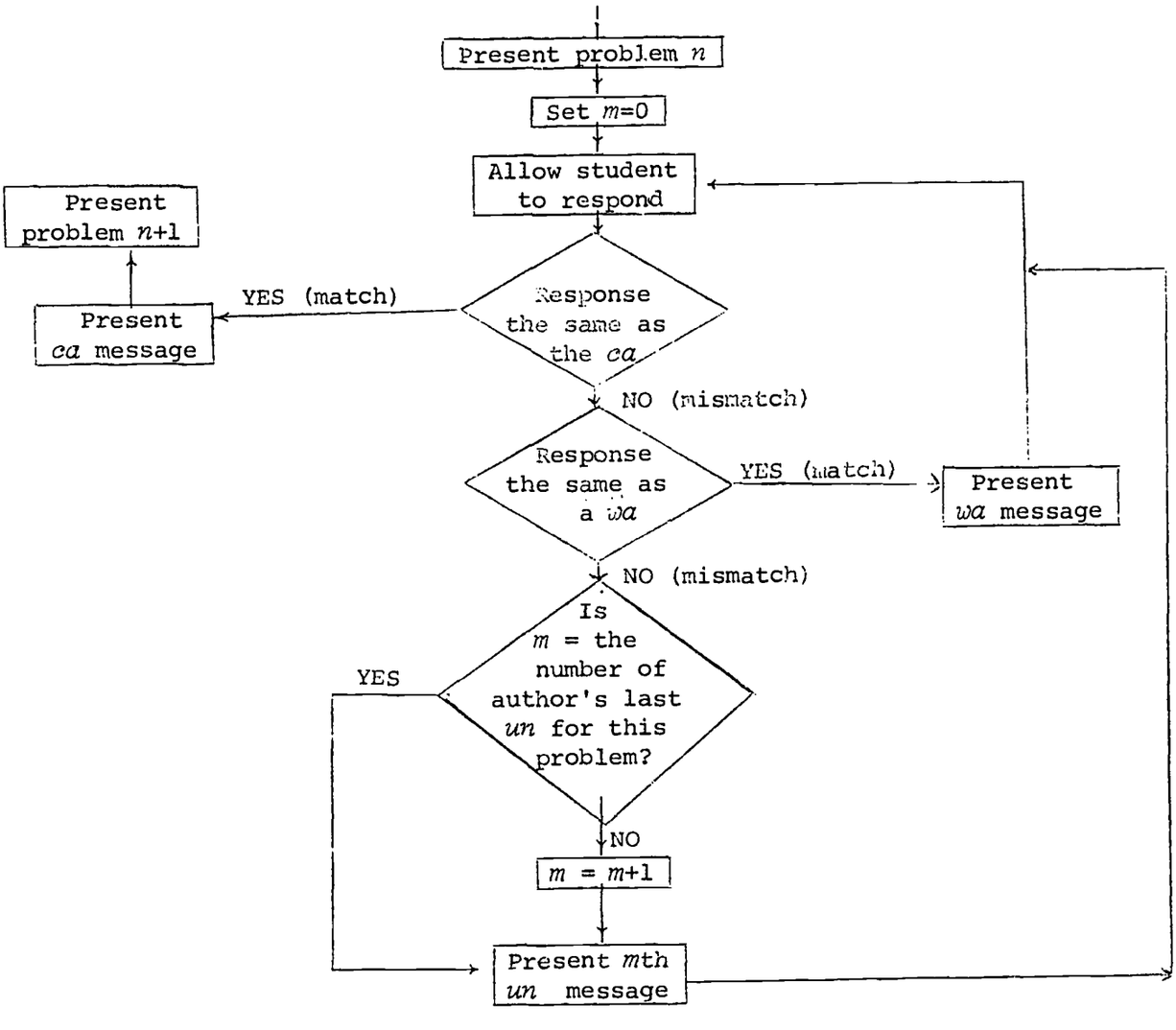
1. *ca*: If the student's entry is the same as this one, present this message to him and then present the next problem. This dependent action (logic) can be described by the following excerpt from a flowchart:



2. *wa*: If the student's entry is the same as this one, present this message to him and then allow him to respond to this question again.



3. *un*: If the student's entry is not the same as any of those expected by the author, present this message to him, and then allow him to respond to this question again. An author may prepare as many as ten *un* messages for any one particular problem. The combined logics--*ca*, *wa*, *un*--are described by the following flowchart:



Keep in mind that an author can use any dependent actions (logic) he wants. That which is described above is but one example.

4. *label:* A label may be up to a maximum of six characters, either alphabetic or numeric, and is used to give sequences of computer code an identity. Labels are internal to the computer and not visible to students. Labels are directly addressible by name by the computer.

5. *epid:* An epid may be up to a maximum of ten characters and is used to give student responses an identity. If epid's and labels are coordinated, then a particular student response can be related to a particular sequence of computer code. Epid's are internal to the computer, not visible to a student, but are saved by the computer as part of the student's response record. These identifiers are not addressible by name by the computer.

6. *frame identifier:* A frame identifier may be up to 40 characters, either alphabetic or numeric, but usually the same as the label. These identifiers are used for debugging and updating of the instructional system by the person in charge of student use of the system to identify the sequence of code currently being executed by the computer for a particular student at a particular time. The person in charge can simply look at the student's display to make the identification. Frame identifiers are visible to students, but are not addressible at all by the computer.

7. *match id:* A match id may be up to a maximum of two characters and is saved by the computer along with the student's response; contents of counters, switches, return registers, and epid. Thus, the match id can be used to identify which of the messages prepared by the author was displayed to a particular student at a particular time.

8. *prr:* Often students are not able to complete a course during one session with the computer. Thus, at the end of a session, the computer needs to know where in the course a particular student should resume his work at the beginning of his next session. *prr* is used by the author to indicate these places (restart points) to coders.

9. *pr:* *pr* is used by the author to mark the beginning of a problem.

At this point, Example 1 on the following page should be self-explanatory to the reader if page numbers are carefully observed.

Refer to Example 1 on page 7

An author may wish to define several problems at one time to reduce repetitious tedium. He can do this by replacing the portions of text that vary with column headings. The varying text is then prepared in tabular form. Thus, fixed text need be written only once. Coders and authors can still see the instructional system operate; and while coding time or storage space may not be reduced, coding is less tedious and debugging time is greatly reduced. An example of this appears on page 8.

Refer to Example 2 on page 8

An author's draft prepared in the manner described above will, to some extent provide for the following:

--Author and student alike will see the same display at the time each is doing his work.

--Author will not be tempted to crowd too much text into one display.

--Course can be programmed in any CAI language.

--Author can conduct an off-machine test of the instruction with a few students.

--Programmer can generate the required computer code for implementation without assistance from the author.

--Neither the author nor the programmer need to repeat text which is to be displayed over and over.

--Student practice may be extended without significantly increasing programming time, author time, or disk storage.

--Modifications and updates are easily accomplished.

--Media specialist can prepare associated audio-visuals.

--Publisher can prepare manuals and permanent documentation for users and potential users.

EXAMPLE 1

C0052I (LABEL)

PRR (RESTART POINT)

FRAME IDENTIFIER

0 4 8 12 16 20 24 28 32 36 39

1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11 $[(x - 2y + 3z)^5]^4 =$ C0052IA (epid)

13

15

17

19

21

23

25

27

29

31

Page 1

EXAMPLE 1

C0052I (*LABEL*)

PRR (*RESTART POINT*)

0 4 8 12 16 20 24 28 32 36 39

FRAME IDENTIFIER

1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11 $[(x - 2y + 3z)^5]^4 =$ $(x - 2y + 3z)^{20}$ *CA* *C1*

13 Correct. (*MATCH ID*)

15

17

19

21

23

25

27

29

31

Page 2

EXAMPLE 1

C0052I (*LABEL*)

PRR (*RESTART POINT*)

FRAME IDENTIFIER	0 4 8 12 16 20 24 28 32 36 39		
1	C0052I (one question)		
3 5 7	Use the third law of exponents to rewrite the following expression. All exponents are positive integers.		
9 11	$[(x - 2y + 3z)^5]^4 =$	$(x - 2y + 3z)^{(5)(4)}$	*CA* *C2* (*MATCH ID*)
13 15 17 19 21 23 25 27 29 31	Ok, but we prefer that you carry your work a step further. We accept your response even though $(x - 2y + 3z)^{20}$ is preferable.		

Page 3

EXAMPLE 1

C0052I (LABEL)

PRR (RESTART POINT)

FRAME IDENTIFIER

0 4 8 12 16 20 24 28 32 36 39

1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11 $[(x - 2y + 3z)^5]^4 =$ $x - 2y + 3z^{20}$ *WA* *W1* *(MATCH ID)*

13 Your thought is probably correct, but

15 you have incorrectly represented the

17 given expression. Since the base itself

19 is an expression, it must be enclosed

21 in parentheses. For example,

23

25 $(a+b)^2 = (a+b)(a+b) = a^2 + 2ab + b^2$.

27 Obviously, $(a+b)^2 \neq a+b^2$ if $a \neq 0$.

29

31 Use () to enclose the base.

Page 4

EXAMPLE 1

C0052I (LABEL)

PRR (RESTART POINT)

FRAME IDENTIFIER

0 4 8 12 16 20 24 28 32 36 39

1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11 $[(x - 2y + 3z)^5]^4 =$ [ ] *UN* *U1 (MATCH ID)*

13

15 Incorrect. Check your work carefully

17 and respond again.

19

21

23

25

27

29

31

Page 5

EXAMPLE 1

C0052I (*LABEL*)

PRR (*RESTART POINT*)

0 4 8 12 16 20 24 28 32 36 39

FRAME IDENTIFIER

1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11 $[(x - 2y + 3z)^5]^4 =$ [ ]

UN U2
(MATCH ID)

13

15 Incorrect. If you think of

17 x - 2y + 3z as b,

19 5 as m,

21 4 as n, then you can see that

23

25 $b^{mn} = (x-2y+3z)^{(5)(4)} = (x-2y+3z)^{20}$.

27 Enter $(x - 2y + 3z)^{20}$ now.

29

31

PAGE 6

TABLE

LIST	$\langle e_1 \rangle$	$\langle e_2 \rangle$	$\langle e_3 \rangle$	$\langle e_4 \rangle$	$\langle n_1 \rangle$	$\langle n_2 \rangle$	$\langle n_3 \rangle$
1	$[(x-2y+3z)^5]^4$	$(x-2y+3z)^{20}$	$x-2y+3z^{20}$	$(x-2y+3z)^{(5)(4)}$	$x-2y+3z$	5	4
2	$[(2a+b)^4]^3$	$(2a+b)^{12}$	$2a+b^{12}$	$(2a+b)^{(4)(3)}$	$2a+b$	4	3
3	$[(-3/4)^t]^2$	$(-3/4)^{2t}$	$-3/4^{2t}$	$(-3/4)^{(t)(2)}$	$-3/4$	t	2
4							
5	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							

EXAMPLE 2

C0052I (*LABEL*)

PRR

0 4 8 12 16 20 24 28 32 36 39

(*FRAME IDENTIFIER*)

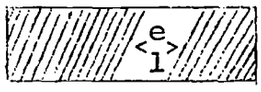
1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11  = C0052IA (epid)

13

15

17

19

21

23

25

27

29

31

PAGE 1

EXAMPLE 2

C0052I (LABEL)

PRR

0 4 8 12 16 20 24 28 32 36 39

(*FRAME IDENTIFIER*)

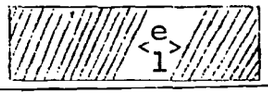
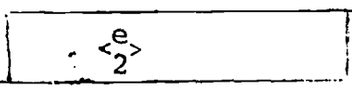
1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11  = 

CA *C1* (*MATCH ID*)

13

15 Correct.

17

19

21

23

25

27

29

31

PAGE 2

EXAMPLE 2

C0052I (LABEL)

PRR

0 4 8 12 16 20 24 28 32 36 39

(FRAME IDENTIFIER)

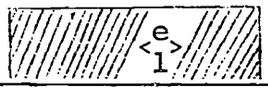
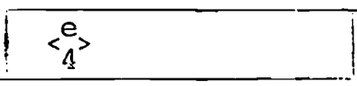
1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11  = 

CA *C2 (MATCH ID)*

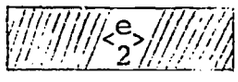
13

15 Ok, but we prefer that you carry your

17 work a step further. We accept your

19

21 response even though

23 

25

27 is preferable.

29

31

PAGE 3

EXAMPLE 2

C0052I (LABEL)

PRR

0 4 8 12 16 20 24 28 32 36 39

(FRAME IDENTIFIER)

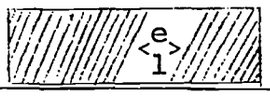
1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11   *WA* *W1* *(MATCH ID)*

13

15 Your thought is probably correct, but

17 you have incorrectly represented the

19 given expression. Since the base itself

21 is an expression, it must be enclosed

23 in parentheses. For example;

25 $(a+b)^2 = (a+b)(a+b) = a^2 + 2ab + b^2$.

27 Obviously, $(a+b)^2 \neq a+b^2$ if $a \neq 0$.

29

31 Use () to enclose the base.

PAGE 4

EXAMPLE 2

C0052I (LABEL)

PRR

0 4 8 12 16 20 24 28 32 36 39

(FRAME IDENTIFIER)

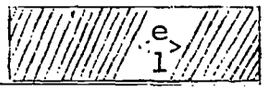
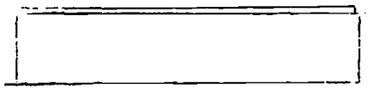
1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11  ⇒ 

UN U1 (MATCH ID)

13

15 Incorrect. Check your work carefully

17 and respond again.

19

21

23

25

27

29

31

PAGE 5

EXAMPLE 2

C0052I (LABEL)

PRR

0 4 8 12 16 20 24 28 32 36 39

(FRAME IDENTIFIER)

1 C0052I (one question)

3 Use the third law of exponents to

5 rewrite the following expression.

7 All exponents are positive integers.

9

11 $\langle e_1 \rangle$ = [ ] *UN* J2 *(MATCH ID)*

13

15 Incorrect. If you think of

17 $\langle n_1 \rangle$ as b,

19

21 $\langle n_2 \rangle$ as m,

23 $\langle n_3 \rangle$ as n, then you can see that

25

27 b^{mn} = $\langle e_4 \rangle$ = $\langle e_2 \rangle$

29 Enter $\langle e_2 \rangle$ now.

31

PAGE 6

The Programmer's Role

Abstract

Our experience with the Preskills and Maths course has led us to believe that close communication between author and coder is essential to producing imaginative yet consistent instructional programming. The traditional roles of an author creating course material and a disassociated coder implementing the creation on a machine were not functional in producing a truly instructional course with expanded answer-processing and a consistent philosophy of the aims of the course. Our aim at The University of Texas at Austin is to make both author and coder aware of the author's designs for the course and to create an author-coder "team" to implement those designs with consistency and imagination.

The Prerequisite Mathematical Skills (Preskills) program is an excellent example of the hazards involved in the traditional authoring-coding methods used by CAI installations. After the course material was authored, a number of coders were employed to produce an operational program. Many coders were involved because of the size of the program and because coders were joining and leaving the staff regularly. The variety of coding styles produced the major defect of the course: a devastating lack of consistency. For a given correct answer of, say, 10000, one part of the course might accept 10,000 or 10000 as a student response, another part might accept only 10000. The limitations placed on a student would depend on the particular coder's dexterity with answer-processing techniques. Furthermore, some coders were content to say only "Wrong. Try again" for an incorrect response, while others would be more extensive in their wrong-answer processing. These inconsistencies were the major complaints of students taking Preskills.

It was our feeling that such inconsistencies stemmed from inadequate author-coder communication. The author, Mrs. Authella Smith, has a definite philosophy of teaching, but this philosophy had failed to be expressed in the Preskills course. It was not enough to have consistent specifications for coding techniques; the lack of consistency in aims and expectations for the course had resulted in the inconsistencies of the final product.

As we began to think of revising Preskills, we were determined to make the course more expressive of the author's philosophy. We favored giving students more control over the material they would take (in response to many complaints from students about being trapped in long sections of material they did not wish to take). We were also very concerned with limitations that the Preskills course had placed on student answer formats. We were determined to expand answer-processing to accept any answer that was correct, and to provide more explicit messages for more types of wrong answers. To improve on the basic machine-student relationship, which hinders many students, we established conventions such as not changing any screen messages until the student indicates that he is ready for a change, and writing messages making it clear to a student that he is dealing with a program written by a human being, not by some super-intelligent electronic "brain."

As these aims became more definite, they produced two effects: First, we turned from the idea of simply revising Preskills to deciding to rewrite the whole course under the name Maths. The second effect was to produce a new type of author-coder relationship in which the coder realizes the aims and intent of the course and works with the author to produce a working expression of the author's teaching philosophy. Only in this way, we felt, could we produce a consistent course in the context of frequent turnover of personnel.

Our new author-coder "team" first approached the idea of student control and produced a table of contents format by which the student selects an area of study but is free to quit and choose another at any time. Diagnostic tests to determine what instruction was needed became evaluations for the student's personal benefit. A glossary was included for student reference from any point in the program.

Again, it was not our purpose to specify techniques, but to standardize the aims of the course. Coding techniques would be the products of the course philosophy, making the course "coder independent" in a new sort of way. After the author-coder "team" specifies what the course should be able to do, implementation procedures should no longer be critical.

The effectiveness of the "team concept" at The University of Texas at Austin is still open to question. In the early development of Maths, a close author-coder relationship was maintained, producing consistent course material using varied techniques that often extended beyond standard coursewriter methods (extensive editing procedures in answer-processing, production of an arithmetic evaluation function). But the small staff underwent extensive turnovers of personnel, and it was greatly enlarged to meet contract commitments. With many new coders coming under emergency conditions, author-coder contact was greatly diminished, and we were left with the same problem faced in the development of the original Preskills course: managing a large, new staff. The needs of the control program, the course philosophy, and

techniques used for past problems proved to be too difficult to teach the new staff in the time available, since students were already committed to the course and production was imperative. As a result, both consistency and innovation were hampered, though not so much as in the original version.

It is still our belief, however, that the programmer's role in CAI lies in a close relationship with the author. As we begin development of a new CAI language, we have a staff familiar with the limitations of Coursewriter and conscious of the requirements of an instructional course. To create a new language, it will be essential to maintain the communication of requirements and new techniques and the feedback of experiences.